ARTI BHATIA

CS5A, 16

**FILE ALLOCATION**

Q- Simulate the following file allocation strategies.

a) Sequential

b) Indexed

c) Linked

**Sequential**

Program:

#include <stdio.h>

int memory[100], directoryTable[100][2], fileNo = 0;

const int mem\_size = 100;

void addFile()

{

    int file\_size, isSpace;

    printf("Size of file: ");

    scanf("%d", &file\_size);

    for (int i = 0; i < mem\_size; i++)

    {

        isSpace = 1;

        for (int j = 0; j < file\_size; j++)

        {

            if (memory[i + j] != -1 || i + j >= mem\_size)

            {

                isSpace = 0;

                break;

            }

        }

        if (isSpace == 1)

        {

            for (int k = 0; k < file\_size; k++)

                memory[i + k] = fileNo;

            directoryTable[fileNo][0] = i;

            directoryTable[fileNo][1] = file\_size;

            fileNo++;

            isSpace = 2;

            break;

        }

    }

    if (isSpace == 2)

        printf("File is allocated to memory successfully");

    else

        printf("Contiguous memory of file size does not exist");

}

void showMemory()

{

    printf("\nMemory\n");

    for (int i = 0; i < mem\_size; i++)

    {

        printf("%d\t", memory[i]);

        if (i % 10 == 9 && i != 0 && i != mem\_size - 1)

            printf("\n");

    }

}

void showDirectoryTable()

{

    printf("\nFile no \tStart Block\tFile Size");

    for (int i = 0; i < fileNo; i++)

    {

        printf("\n   %d\t\t     %d\t\t    %d", i, directoryTable[i][0], directoryTable[i][1]);

    }

}

void main()

{

    //initialize memory

    for (int i = 0; i < mem\_size; i++)

        memory[i] = -1;

    printf("\n----------------SEQUENTIAL FILE ALLOCATION----------------\n");

    printf("\n1.Add File");

    printf("\n2.Display Memory");

    printf("\n3.Display Directory Table");

    printf("\n4.Exit");

    int ch;

    do

    {

        printf("\n\nYour choice: ");

        scanf("%d", &ch);

        switch (ch)

        {

        case 1:

            addFile();

            break;

        case 2:

            showMemory();

            break;

        case 3:

            showDirectoryTable();

            break;

        case 4:

            break;

        default:

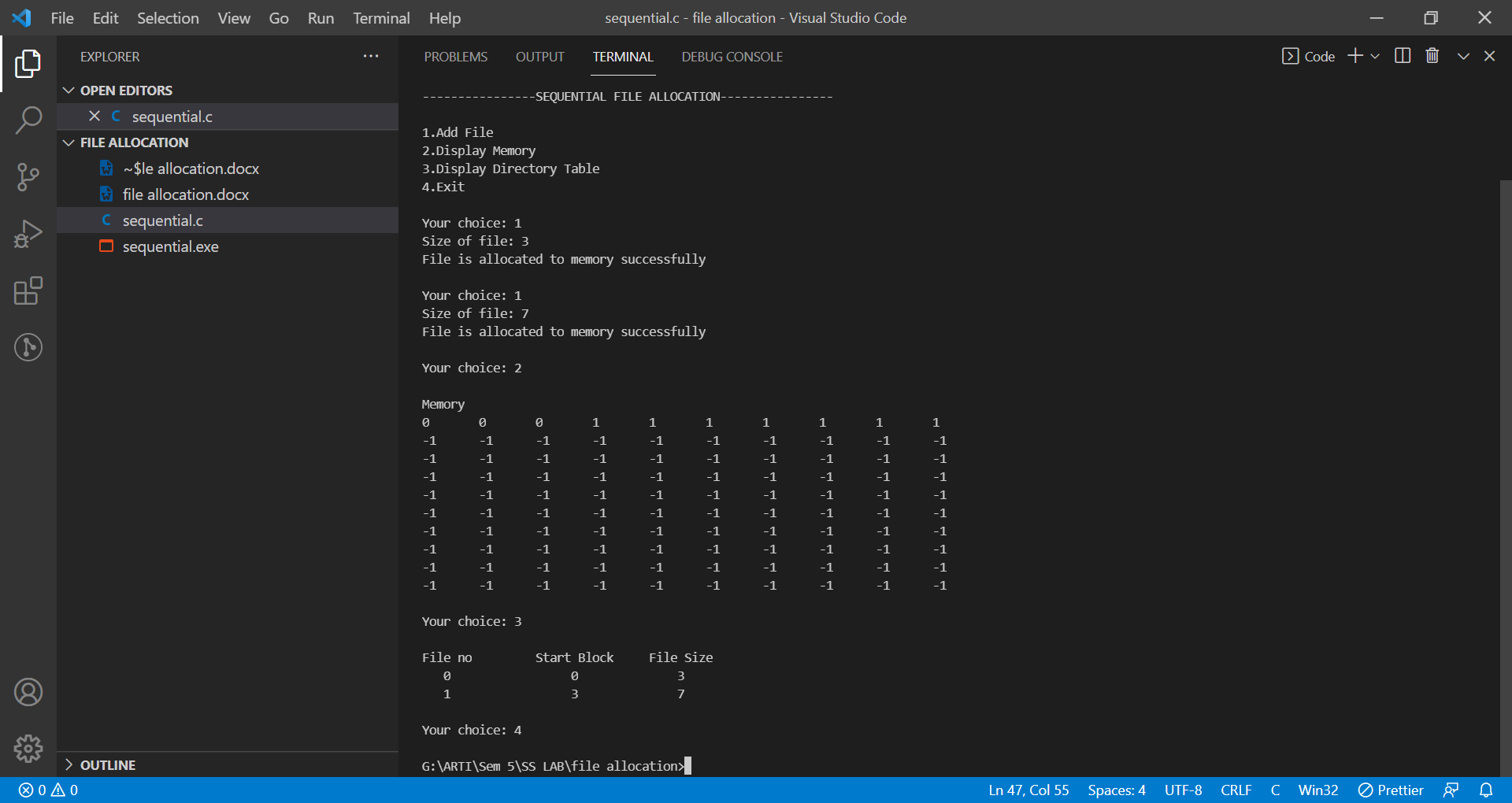
            printf("\nWrong Input");

        }

    } while (ch != 4);

}

Output:



**Indexed**

Program:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

struct block

{

    int data;

    int ptr[99];

} memory[100];

int directoryTable[100], fileNo = 0, remainingSpace = 100;

const int mem\_size = 100;

void addFile()

{

    int file\_size, isSpace;

    printf("Size of file: ");

    scanf("%d", &file\_size);

    if (file\_size > remainingSpace)

        printf("File Size is more than the remaining space in the memory");

    srand(time(0));

    //chosing a block to act as index table

    int index\_block = rand() % mem\_size;

    memory[index\_block].data = -2;

    remainingSpace--;

    int count = 0;

    while (count != file\_size)

    {

        int location = rand() % mem\_size;

        if (memory[location].data == -1)

        {

            memory[location].data = fileNo;

            memory[index\_block].ptr[count] = location;

            if (count == 0)

                directoryTable[fileNo] = index\_block;

            remainingSpace--;

            count++;

        }

    }

    fileNo++;

    printf("File allocated successfully");

}

void showMemory()

{

    printf("\nMemory\n");

    for (int i = 0; i < mem\_size; i++)

    {

        printf("%d\t", memory[i].data);

        if (i % 10 == 9 && i != 0 && i != mem\_size - 1)

            printf("\n");

    }

}

void showDirectoryTable()

{

    printf("\nFile no \tIndex Table Location");

    for (int i = 0; i < fileNo; i++)

    {

        printf("\n   %d\t\t\t%d", i, directoryTable[i]);

    }

}

void showLinkedList()

{

    printf("\nIndex:\n");

    for (int i = 0; i < fileNo; i++)

    {

        int IndexTableLoc = directoryTable[i], j = 0, currIndexValue = 0;

        while (currIndexValue != -1)

        {

            currIndexValue = memory[IndexTableLoc].ptr[j];

            printf("%d", currIndexValue);

            j++;

            if (currIndexValue != -1)

                printf(" -> ");

        }

        printf("\n");

    }

}

void main()

{

    //initialize memory

    for (int i = 0; i < mem\_size; i++)

    {

        memory[i].data = -1;

        for (int j = 0; j < mem\_size - 1; j++)

            memory[i].ptr[j] = -1;

    }

    printf("\n----------------INDEXED FILE ALLOCATION----------------\n");

    printf("\n1.Add File");

    printf("\n2.Display Memory");

    printf("\n3.Display Directory Table");

    printf("\n4.Display Index");

    printf("\n5.Exit");

    int ch;

    do

    {

        printf("\n\nYour choice: ");

        scanf("%d", &ch);

        switch (ch)

        {

        case 1:

            addFile();

            break;

        case 2:

            showMemory();

            break;

        case 3:

            showDirectoryTable();

            break;

        case 4:

            showLinkedList();

            break;

        case 5:

            break;

        default:

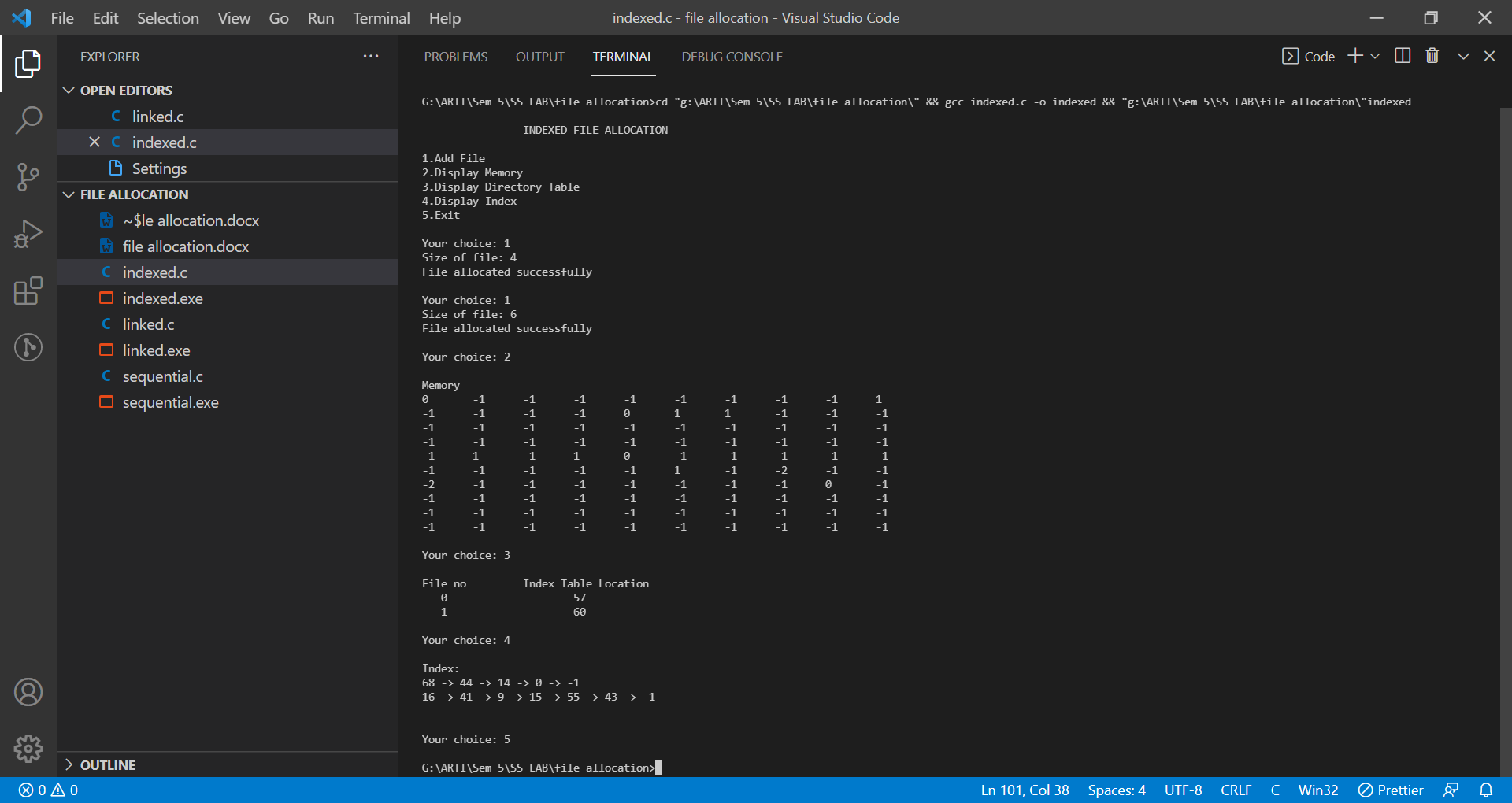
            printf("\nWrong Input");

        }

    } while (ch != 5);

}

Output:



**Linked**

Program:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

struct block

{

    int data;

    int ptr;

} memory[100];

int directoryTable[100][2], fileNo = 0, remainingSpace = 100;

const int mem\_size = 100;

void addFile()

{

    int file\_size, isSpace;

    printf("Size of file: ");

    scanf("%d", &file\_size);

    if (file\_size > remainingSpace)

        printf("File Size is more than the remaining space in the memory");

    srand(time(0));

    int prevBlock = -1, count = 0;

    while (count != file\_size)

    {

        int location = rand() % mem\_size;

        if (memory[location].data == -1)

        {

            memory[location].data = fileNo;

            remainingSpace--;

            if (count == 0)

                directoryTable[fileNo][0] = location;

            else if (count == file\_size - 1)

                directoryTable[fileNo][1] = location;

            if (prevBlock != -1)

            {

                memory[prevBlock].ptr = location;

                prevBlock = location;

            }

            else

                prevBlock = location;

            count++;

        }

    }

    fileNo++;

    printf("File allocated successfully");

}

void showMemory()

{

    printf("\nMemory\n");

    for (int i = 0; i < mem\_size; i++)

    {

        printf("%d\t", memory[i].data);

        if (i % 10 == 9 && i != 0 && i != mem\_size - 1)

            printf("\n");

    }

}

void showDirectoryTable()

{

    printf("\nFile no \tStart Block\tEnd Block");

    for (int i = 0; i < fileNo; i++)

    {

        printf("\n   %d\t\t     %d\t\t    %d", i, directoryTable[i][0], directoryTable[i][1]);

    }

}

void showLinkedList()

{

    printf("\nLinked List:\n");

    for (int i = 0; i < fileNo; i++)

    {

        int pointer = directoryTable[i][0];

        while (pointer != -1)

        {

            printf("%d", pointer);

            pointer = memory[pointer].ptr;

            if (pointer != -1)

                printf(" -> ");

        }

        printf("\n");

    }

}

void main()

{

    //initialize memory

    for (int i = 0; i < mem\_size; i++)

    {

        memory[i].data = -1;

        memory[i].ptr = -1;

    }

    printf("\n----------------LINKED FILE ALLOCATION----------------\n");

    printf("\n1.Add File");

    printf("\n2.Display Memory");

    printf("\n3.Display Directory Table");

    printf("\n4.Display Linked List");

    printf("\n5.Exit");

    int ch;

    do

    {

        printf("\n\nYour choice: ");

        scanf("%d", &ch);

        switch (ch)

        {

        case 1:

            addFile();

            break;

        case 2:

            showMemory();

            break;

        case 3:

            showDirectoryTable();

            break;

        case 4:

            showLinkedList();

            break;

        case 5:

            break;

        default:

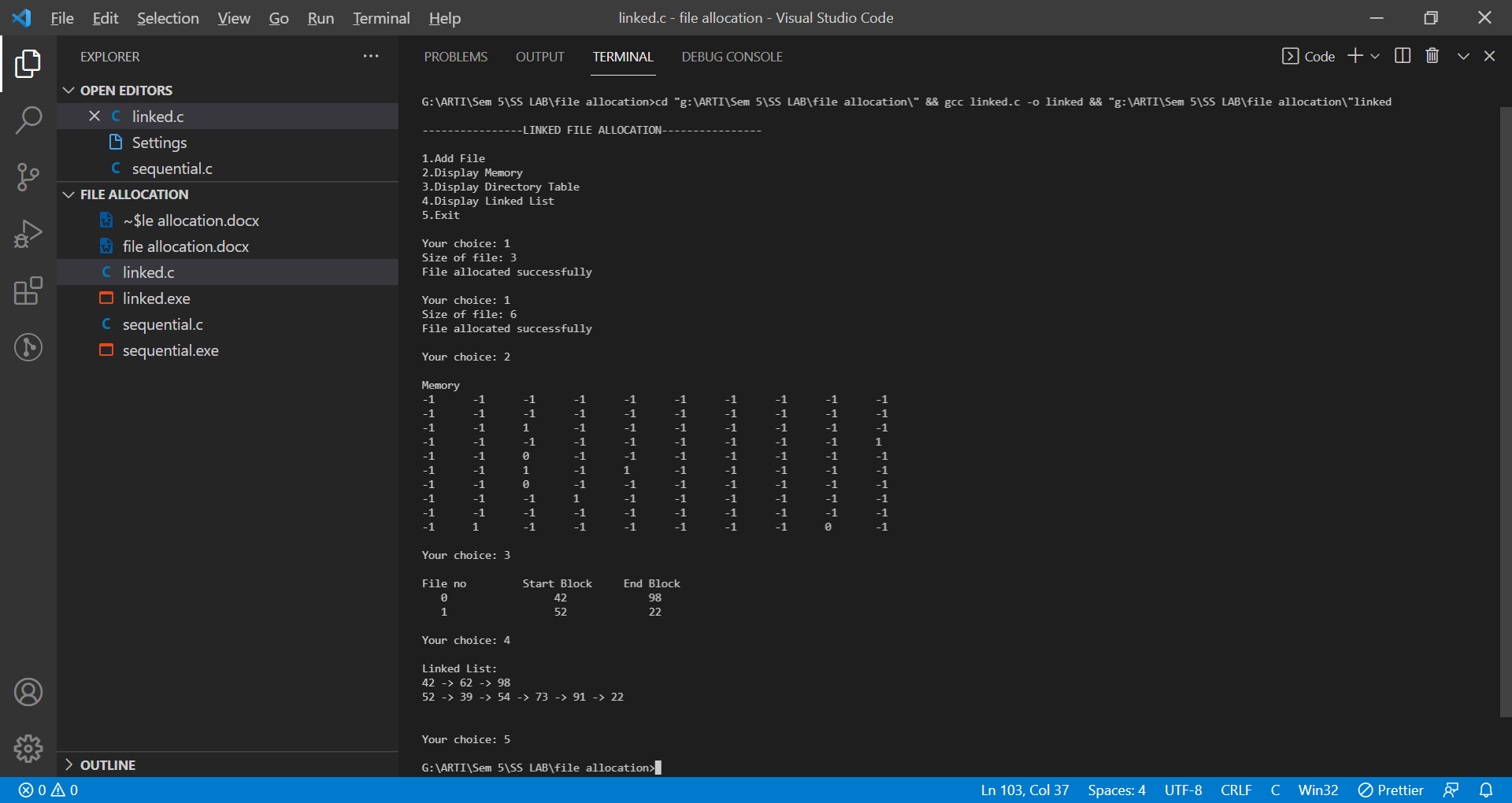
            printf("\nWrong Input");

        }

    } while (ch != 5);

}

Output:



**ALGORITHMS**

